

Linguagem XSD (“XML Schema Definition”)

O que é o XSD?

- É uma linguagem de base XML para especificar estrutura (esquema)
 - de outros documentos XML (instâncias do esquema)
- XSD e DTD (“Document Type Definition”) têm o mesmo objectivo
 - ... o de especificar os blocos de construção válidos num documento XML
- O XSD é uma alternativa ao DTD
 - ... o XSD permite especificar tudo o que o DTD permite e bastante mais!
- Alguns aspectos que contribuem para o interesse do XSD
 - suporta tipos de dados e derivação (o DTD apenas suporta texto)
 - suporta integração com espaços de nome (o DTD não suporta)
- ... e é escrito em XML (o DTD tem uma sintaxe própria)
 - ... assim, a tecnologia XML aplica-se toda ao XSD (e.g XPath, XQuery)
 - ganho muito importante: tratamento uniforme da estrutura e dos dados...

O que são: Esquema e Instância?

- Um esquema é uma especificação formal
 - da estrutura que um documento válido tem que respeitar
- Um documento é instância de um esquema
 - quando a sua estrutura é descrita por esse esquema
 - ... diz-se documento instância ou simplesmente instância
- A construção de um esquema corresponde a
 - escrever um (ou mais) ficheiros XSD
- A construção de uma instância corresponde a
 - escrever um (ou mais) ficheiros XML associado(s) a ficheiro(s) XSD

Documento XML: bem formado e válido

- Um documento XML diz-se bem formado
 - se respeitar as regras da linguagem XML
 - e.g. “o valor dos atributos é delimitado por aspas ou plicas”, etc
- Um documento XML diz-se válido de acordo com um esquema
 - ... ou conforme com um esquema (“schema valid”)
 - sempre que respeita (satisfaz) todas as restrições do esquema
 - ... se é válido está bem formado (o inverso pode não ser verdade)
- Note-se que uma especificação XSD é também documento XML
 - pode estar, ou não, bem formado
 - pode estar, ou não, válido (de acordo com a sua especificação XSD!)
- Há ferramentas que verificam um documento XML
 - e.g. o “Notepad XML”, o “XML Spy”, ...
 - analisador na Internet em: <http://www.w3.org/2001/03/webdata/xsv>

Uma convenção (para simplificar o entendimento)

O XSD concretiza-se como um documento XML.

Assim, os acrónimos XSD e XML misturam-se!

Já sabemos isso e queremos separar as coisas, portanto ...

A convenção. Sempre que não exista ambiguidade vamos chamar:

- XML ao documento (instância) que respeita determinado esquema, e
- XSD ao documento onde se especifica um esquema.

Um documento XML que respeita um esquema XSD

O documento XML; ficheiro = receita_v01.xml

```
<?xml version="1.0"?>
```

```
<n:receita xmlns:n="http://EstruturaDasReceitas"
```

"namespace" onde estão declarados os elementos

poderia usar vários "namespace"

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://EstruturaDasReceitas receita_v01.xsd">
```

```
</n:receita>
```

"namespace URI" que coincide com o "targetNamespace" do XSD

localização física do documento XSD

O documento XML acima respeita este esquema XSD; ficheiro = receita_v01.xsd

```
<?xml version="1.0"?>
```

```
<!-- Um "hello world" do XML Schema Language (XSD) -->
```

elemento schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
targetNamespace="http://EstruturaDasReceitas">
```

```
<xs:element name="receita" type="xs:string" />
```

```
</xs:schema>
```

... sem especificação de “namespace”

O documento XML; ficheiro ≡ receita_v01_1.xml

```
<?xml version="1.0"?>
<receita xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="receita_v01_1.xsd">
</receita>
```

XSD não pode definir
“targetNamespace”

localização física
do documento XSD

O documento XML acima respeita este esquema XSD; ficheiro ≡ receita_v01_1.xsd

```
<?xml version="1.0"?>
<!-- Um "hello world" (sem "namespace") do XML Schema Language -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="receita" type="xs:string" />
</xs:schema>
```

só o “namespace”
dos elementos
XSD, e.g. schema

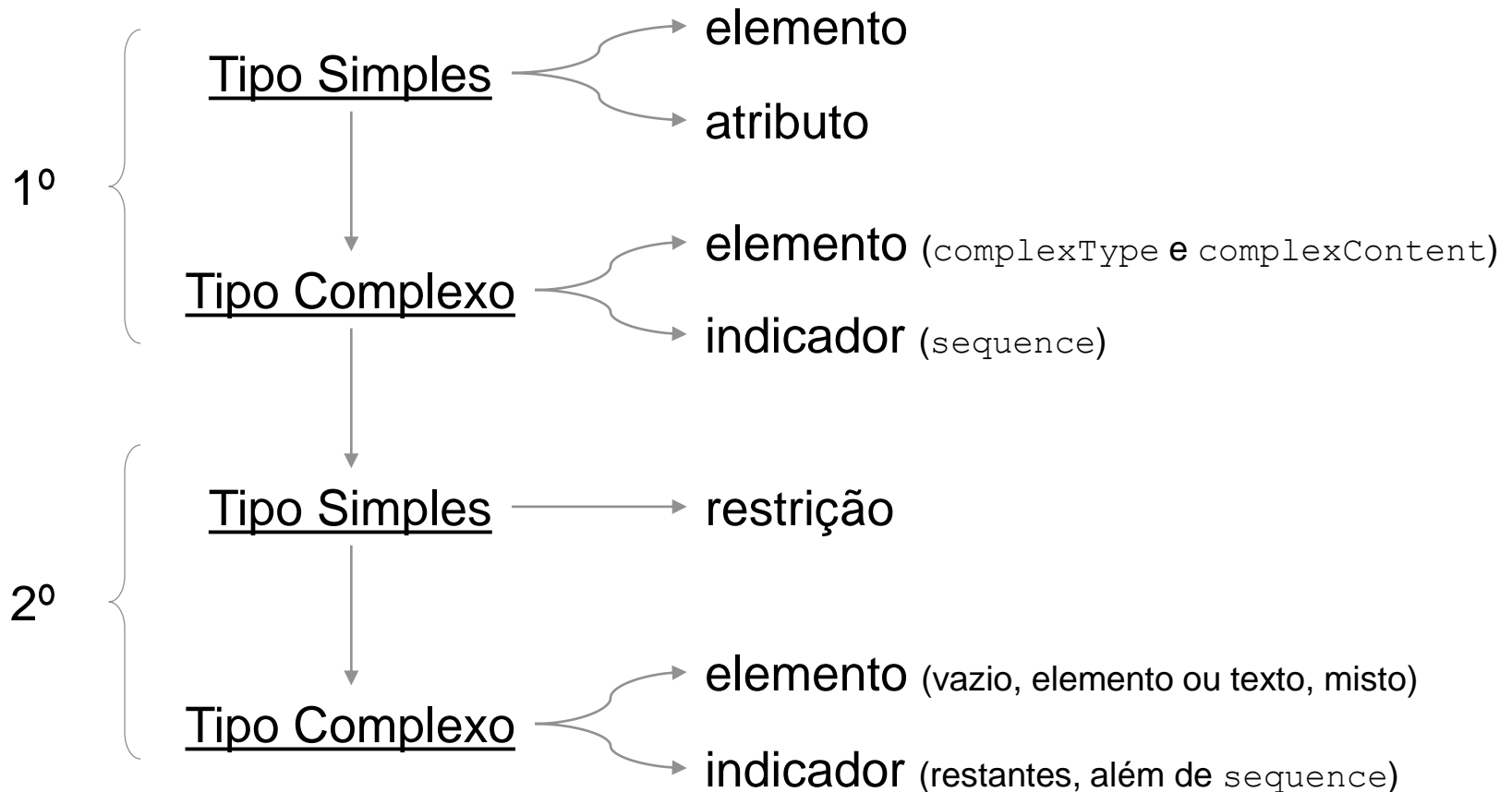
Mas qual o formato geral de um esquema?

- Um esquema (XSD) é um documento XML
 - portanto, tem um elemento raiz e pode ter uma declaração XML
- O elemento raiz de um esquema é o elemento `<schema>`
 - que tem, entre outros, o atributo `targetNamespace`
- Diz-se que um elemento é global quando
 - é descendente imediato do elemento `schema`
- Diz-se que um elemento é local quando
 - é descendente imediato de um elemento diferente de `schema`

... e que estruturas tenho para especificar um esquema?

- Posso especificar um Tipo Simples
 - elemento, atributo e restrição (um novo conceito XSD)
- Posso especificar um Tipo Complexo
 - elemento que contém,
 - ◊ outros elementos (ou outros elementos e texto)
 - ◊ ... cada um dos elementos podendo conter também atributos
- ... e um Tipo Complexo pode ter diversos indicadores
 - indicador de relação de ordem
 - ◊ all, choice, sequence
 - indicador de ocorrência
 - ◊ maxOccurs, minOccurs
 - indicador de agrupamento
 - ◊ group name, attributeGroup name

... sequência desta apresentação ...



Tipo Simples: elemento

- Um elemento de Tipo Simples apenas pode conter texto
 - não pode conter outros elementos nem atributos
 - ... embora o texto possa ser de diferentes tipos

- Um elemento define-se com a sintaxe (simplificada),

```
<xs:element name="xxx" type="yyy" />
```

xs:string
xs:decimal
xs:integer
xs:boolean
xs:date
xs:time
...

- Exemplo,

```
<primNome>Miguel</primNome>  
<idade>10</idade>  
<dataNasc>1997-03-27</dataNasc>
```

```
<xs:element name="primNome" type="xs:string" />  
<xs:element name="idade" type="xs:integer" />  
<xs:element name="dataNasc" type="xs:date" />
```

elementos
XML

definições
XSD

Tipo Simples: elemento (sintaxe mais completa)

<xs:element

name = "XMLname"

type = "QName">

default = "string"

fixed = "string"

nillable = "(true | false)"

block = "(#all | extension | restriction | substitution)"

maxOccurs = "(nonNegativeInteger | unbounded)"

minOccurs = "nonNegativeInteger"

ref = "QName"

substitutionGroup = "QName"

</xs:element>

QName ≡ "Qualified Name"
(nome qualificado)

valor omissão

valor fixo

valor nulo

não permite a
construção
indicada
(#all ≡ não
permite
nenhuma)

restrições

referência ao elemento de uma outra declaração de topo, e
substituição de elementos (a ver posteriormente)

... exemplo (o XSD e o XML) de tipo simples: elemento

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="lista_dataNascimento">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="dataNascimento" type="xs:date"
          nillable="true"
          minOccurs="1"
          maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

exemplo_elemento_v01.xsd

especificação do
elemento
(o resto será visto
posteriormente)

... e um documento XML

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<lista_dataNascimento xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="exemplo_elemento_v01.xsd">
  <dataNascimento>2007-01-31</dataNascimento>
  <dataNascimento xsi:nil="true"></dataNascimento>
</lista_dataNascimento>
```

elemento nulo ≠ elemento vazio

≠
elemento vazio ≡ <dataNascimento/>

Tipo Simples: atributo

- Um atributo é sempre declarado como Tipo Simples
- ... recordar: um elemento de Tipo Simples não pode ter atributos
 - se tiver atributos é considerado um Tipo Complexo

- Um atributo define-se com a sintaxe (simplificada),

```
<xs:attribute name="xxx" type="yyy" />
```

xs:string
xs:decimal
xs:integer
xs:boolean
xs:date
xs:time
...

- Exemplo,

```
<primNome lingua="PT">Miguel</primNome>
```

elementos
XML

```
<xs:attribute name="lingua" type="xs:string" />
```

definições
XSD

Tipo Simples: atributo (sintaxe mais completa)

Sintaxe idêntica à do elemento, no entanto com menos atributos de configuração.

```
<xs:attribute
```

```
  name = "XMLname"
```

```
  type = "QName">
```

```
  default = "string"
```

```
  fixed = "string"
```

```
  use = "(optional | prohibited | required)"
```

```
  ref = "QName"
```

```
</xs:attribute>
```

valor omissão

valor fixo

valor opcional,
proibido ou
obrigatório

referência ao atributo de uma outra declaração de topo

... exemplo (o XSD e o XML) de tipo simples: atributo

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:attribute name="data" type="xs:date"/>
  <xs:element name="lista_dataNascimento">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="dataNascimento" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute ref="data" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Exemplo em que a declaração de um atributo refere (ref) uma declaração de topo.

especificação do atributo
(o resto será visto posteriormente)

... e um documento XML

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<lista_dataNascimento xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="exemplo_atributo_v01.xsd">
  <dataNascimento data="2007-01-31"/>
  <dataNascimento data="2000-12-31"/>
</lista_dataNascimento>
```

atributo é obrigatório

Tipo de dados: `string` (e tipos derivados)

- `string` pode conter
 - qualquer alfanumérico, tabulador, fim de linha e mudança de linha
- `normalizedString` (deriva de `string`)
 - elimina carácter tabulador + fim de linha + mudança de linha
- `token` (deriva de `string`)
 - ... elimina anteriores + múltiplos espaços + espaços iniciais e finais
- Restrições sobre `string`
 - `enumeration`,
 - `length`, `maxLength`, `minLength`, `pattern`,
 - `whiteSpace`

... posteriormente se
verá como concretizar a
derivação de tipos...

Tipo de dados: date, time, dateTime e period

- `date` pode conter uma data
 - no formato: `YYYY-MM-DD` (todos obrigatórios)
- `time` pode conter um instante dentro das 24 horas do dia
 - no formato: `hh:mm:ss` (todos obrigatórios)
- `dateTime` pode conter um instante de tempo
 - formato: `YYYY-MM-DDThh:mm:ss` (todos obrigatórios)
 - e.g. `<inicioAlmoco>2007-01-01T13:30:00</inicioAlmoco>`
- `period` pode conter uma duração de tempo (`n` é um natural)
 - formato: `PnY-nM-nDTnH:nM:nS` (todos obrigatórios)
 - e.g. `<cozedura>P0Y-0M-0DT00:30:00</cozedura>` (meia hora)
- Restrições sobre estes tipos de dados
 - `enumeration`, `maxExclusive`, `maxInclusive`,
 - `minExclusive`, `minInclusive`, `pattern`, `whiteSpace`

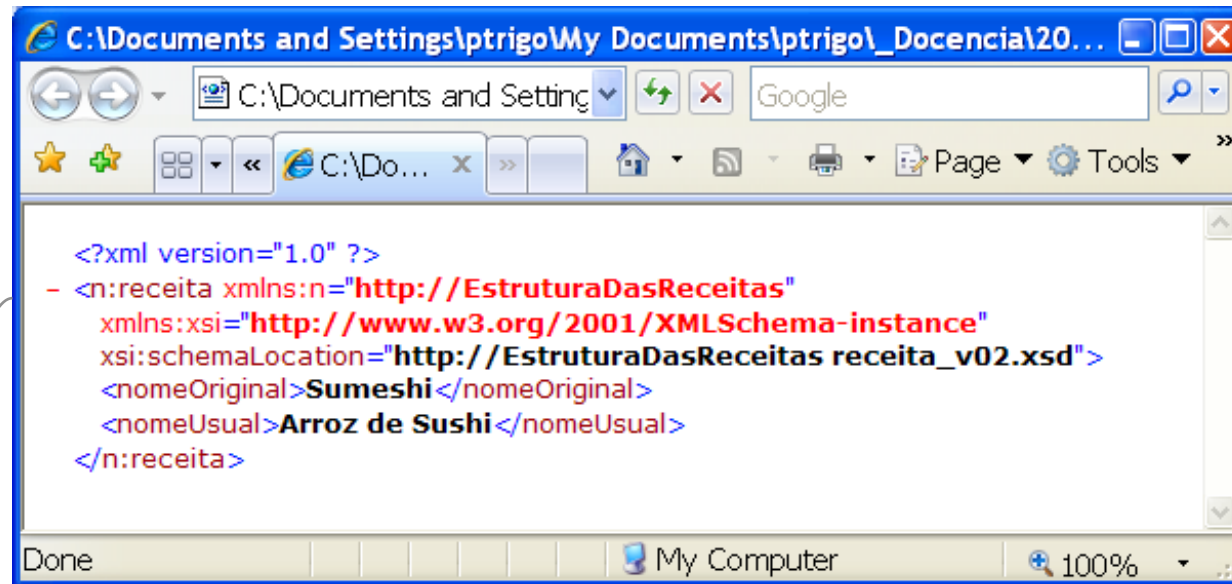
Tipo de dados: decimal (e tipos derivados)

- decimal pode conter numérico (máximo 18 dígitos decimais)
 - e.g. <precoSaldo>29.50</precoSaldo>
 - e.g. <diferenca>-2.50</diferenca>
- integer pode conter inteiro (deriva de decimal)
 - e.g. <idade>29</idade>
 - e.g. <diferenca>-2</diferenca>
- ... outros tipos derivados de decimal
 - byte, long (*signed 64 bits integer*), negativeInteger, nonNegativeInteger, positiveInteger, nonPositiveInteger
- Restrições sobre estes tipos de dados
 - enumeration, fractionDigits, maxExclusive, maxInclusive,
 - minExclusive, minInclusive, pattern, totalDigits, whiteSpace

... outros tipos de dados

- **boolean** pode conter **true** ou **false**
 - e.g. `<xs:attribute name="emVigor" type="xs:boolean"/>`
 - e.g. `<precoSaldo emVigor="false">29.50</precoSaldo>`
- **anyURI** pode conter um qualquer URI
 - e.g. `<xs:attribute name="imagem" type="xs:anyURI"/>`
 - e.g. `<precoSaldo imagem="./a.gif">29.50</precoSaldo>`
 - ... nota: num URI os espaços em branco representam-se por %20
- **QName** pode conter um qualquer nome qualificado
 - e.g. `<xs:attribute name="teste" type="xs:QName"/>`
 - e.g. `<elemento teste="umNomeQualificado"/>`
 - ... notar que um nome qualificado pode, ou não, ser prefixado:
 - ◊ se for prefixado, então o prefixo está associado a “namespace”
 - ◊ se não for prefixado, então o nome ocorre no âmbito local

Tipo Complexo: o nome original e usual de uma receita!



Um Tipo Complexo
composto por dois
Tipos Simples

receita

nomeOriginal

nomeUsual

Tipo Simples

Tipo Complexo

- Um elemento de Tipo Complexo pode conter
 - outros elementos e/ou atributos

```
<receita>
  <nomeOriginal>Sumeshi</nomeOriginal>
  <nomeUsual>Arroz de Sushi</nomeUsual>
</receita>
```

receita_v02.xml

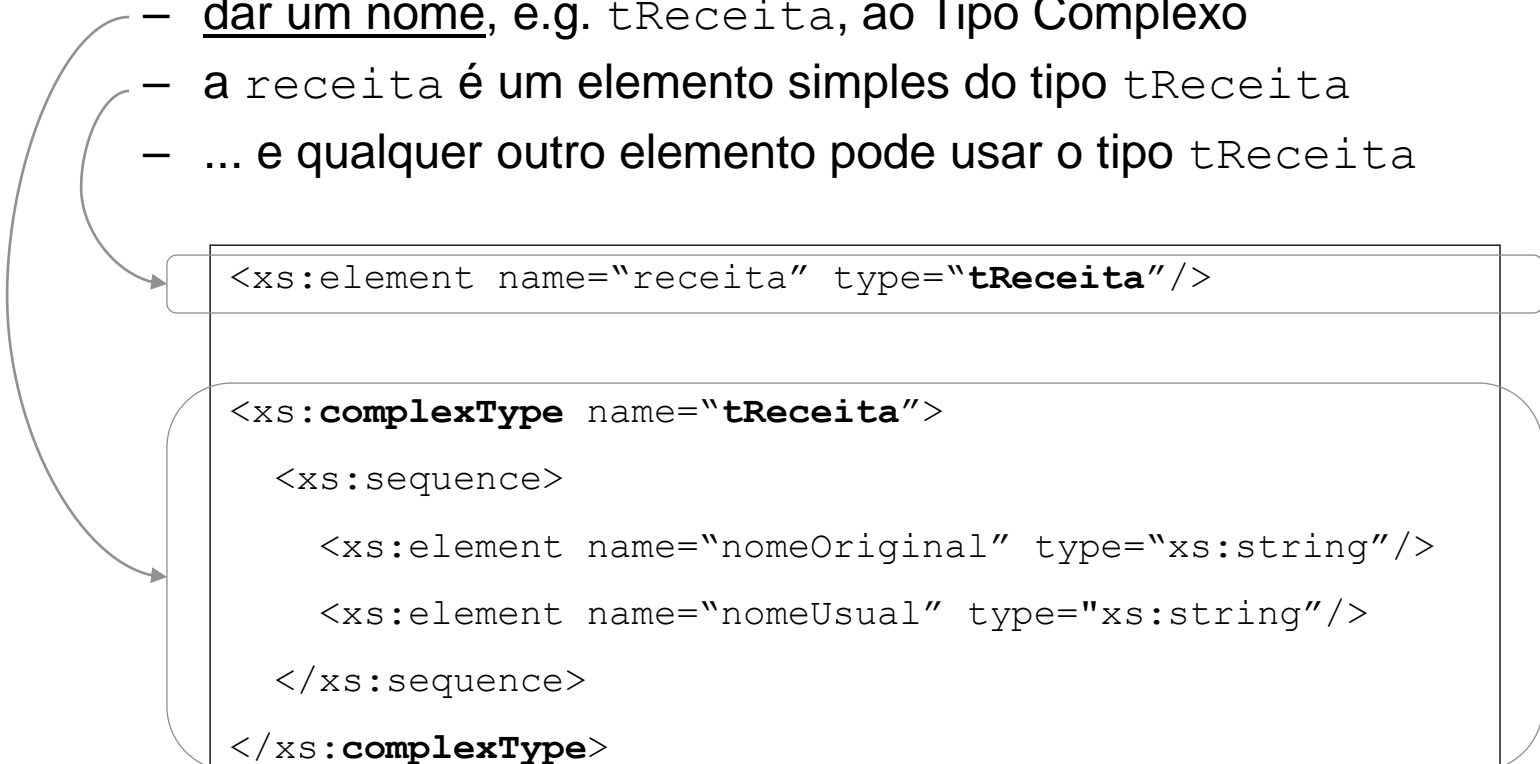
```
<xs:element name="receita">
  <xs:complexType>
    <xs:sequence>
      1º → <xs:element name="nomeOriginal" type="xs:string"/>
      2º → <xs:element name="nomeUsual" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

elementos filho têm que seguir ordem de declaração

receita_v02.xsd

Tipo Complexo (definir com nome)

- ... limitação da solução anterior: o Tipo Complexo era anônimo
 - apenas o elemento `receita` podia usar o Tipo Complexo especificado
- Outra solução
 - dar um nome, e.g. `tReceita`, ao Tipo Complexo
 - a `receita` é um elemento simples do tipo `tReceita`
 - ... e qualquer outro elemento pode usar o tipo `tReceita`

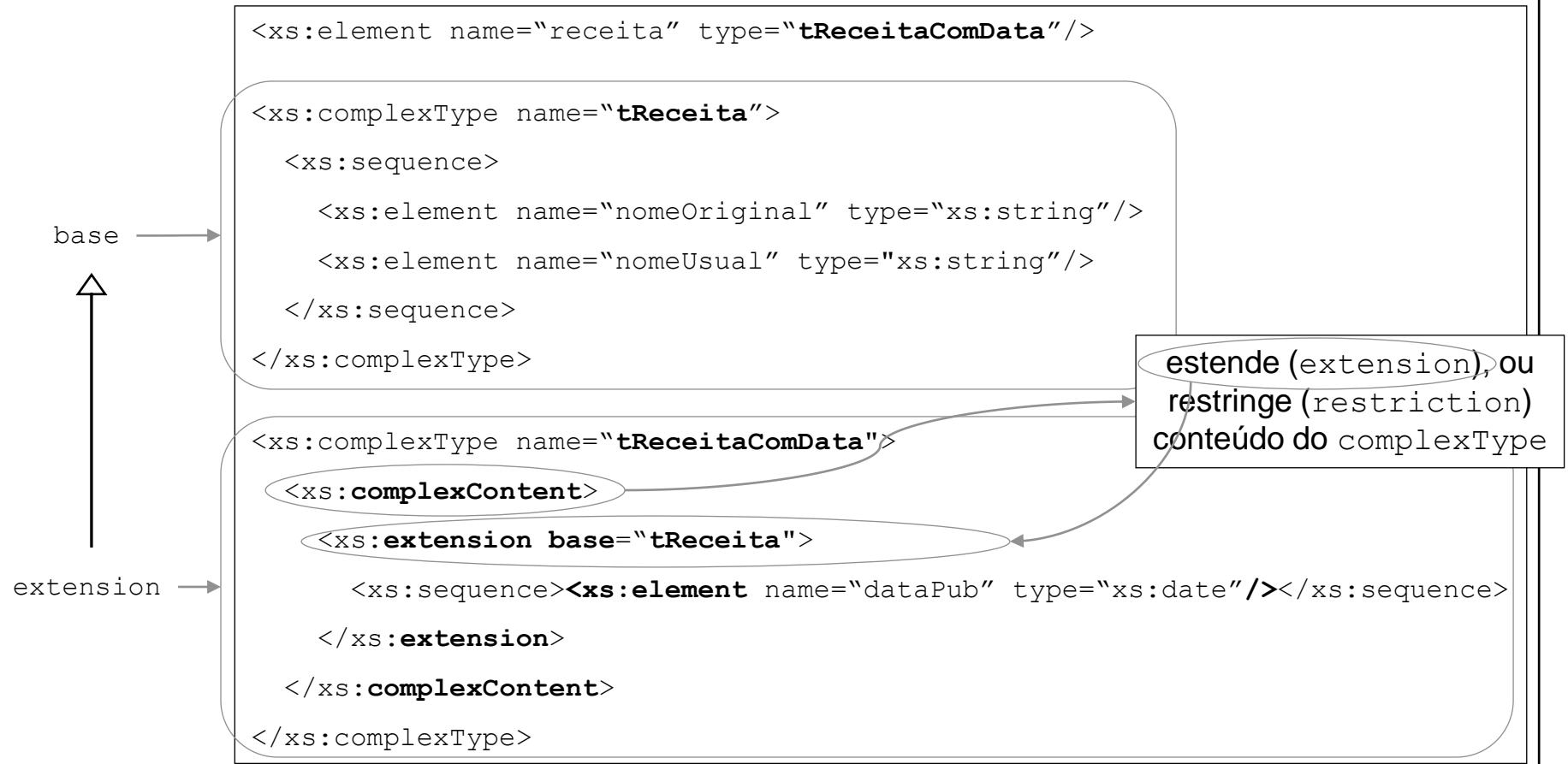


```
<xs:element name="receita" type="tReceita"/>
```

```
<xs:complexType name="tReceita">  
  <xs:sequence>  
    <xs:element name="nomeOriginal" type="xs:string"/>  
    <xs:element name="nomeUsual" type="xs:string"/>  
  </xs:sequence>  
</xs:complexType>
```

Tipo Complexo estendido

- A boa prática da reutilização por derivação consegue-se
 - estendendo um Tipo Complexo (e.g. para conter um novo elemento)



... e novamente o Tipo Simples: a noção de restrição

- Um Tipo Simples (elemento ou atributo) pode ter valores
 - que por sua vez podem estar sujeitos a restrições
 - e.g. a “idade” tem valor inteiro compreendido entre 0 e 130 (será?!)
- Uma restrição define os valores admissíveis para
 - elementos e atributos
- Chama-se faceta (“facet”)
 - a uma restrição definida sobre um elemento
- Uma restrição é sempre um novo Tipo Simples
 - ou seja, não corresponde nem a um elemento nem a um atributo
 - ... é algo que se pretende impor a um elemento ou a um atributo
- Uma restrição pode referir-se a
 - intervalo de valores, conjunto de valores e padrão de valores

Tipo Simples: faceta (intervalo de valores)

restrição

O elemento `idade` tem valor no intervalo de 0 a 120.

... recordar:
faceta é uma
restrição sobre
um elemento

instância
válida

`<idade>10</idade>`

esquema

```
<xs:element name="idade">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

uma restrição é sempre
um novo Tipo Simples

idêntico a estender um tipo base

aqui usa-se `restriction`
para estender usa-se `extension`

Tipo Simples: faceta (conjunto de valores)

O elemento `iberico` apenas pode tomar os valores Portugal e Espanha.

```
<iberico>Portugal</iberico>
```

Outra solução
... em que difere da anterior?

```
<xs:element name="iberico">

  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Portugal"/>
      <xs:enumeration value="Espanha"/>
    </xs:restriction>
  </xs:simpleType>

</xs:element>
```

```
<xs:element name="iberico" type="tIberico"/>

<xs:simpleType name="tIberico">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Portugal"/>
    <xs:enumeration value="Espanha"/>
  </xs:restriction>
</xs:simpleType>
```

Tipo Simples: faceta (padrão de valores)

O elemento `acronimo` tem sempre três letras maiúsculas.

```
<acronimo>IVA</acronimo>
```

```
<xs:element name="acronimo">

  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z] [A-Z] [A-Z]" />
    </xs:restriction>
  </xs:simpleType>

</xs:element>
```

zero ou mais; e.g. minúsculas...
um ou mais; e.g. sToP...
precisamente *n*; e.g. *password*...
este ou este ou este; e.g. um carácter...
este ou este ou este; e.g. o género...

```
... <xs:pattern value="([a-z])*" />
... <xs:pattern value="([a-z][A-Z])+"/>
... <xs:pattern value="([a-zA-Z0-9]){8}" />
... <xs:pattern value="([xyz])+"/>
... <xs:pattern value="masc|femin" />
```

Tipo Simples: faceta (sobre espaços em branco)

O elemento `morada` mantém espaços.

*The **white space** characters are:
spaces, line feeds, tabs and carriage returns.*

```
<morada>Rua  de ...</morada>
```

```
<xs:element name="morada">

  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="preserve"/>
    </xs:restriction>
  </xs:simpleType>

</xs:element>
```

Todos os “white spaces” são substituídos por caracteres de espaço.

```
... <xs:whiteSpace value="replace"/>
```

Múltiplos espaços, e restantes “white spaces”, substituídos por um único carácter de espaço.

```
... <xs:whiteSpace value="collapse"/>
```

Tipo Simples: faceta (sobre número de caracteres)

O elemento `senha` tem entre 5 e 8 caracteres.

```
<senha>aSen5ha</senha>
```

```
<xs:element name="senha">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Se forem exigidos precisamente 4 caracteres (e.g. código multibanco).

```
... <xs:length value="4"/>
```

Mas o código do telemóvel é constituído por 4 dígitos (0 a 9).

```
<xs:element name="codATM">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:totalDigits value="4"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Formas de Elemento de Tipo Complexo

- Um elemento de Tipo Complexo pode conter
 - outros elementos e/ou atributos
- Existem 4 formas de elemento de Tipo Complexo
 - elemento-vazio (que apenas pode conter atributos)
 - elemento-texto (elemento que apenas pode conter texto)
 - elemento-elementos (que apenas pode conter outros elementos)
 - elemento-misto (que pode conter outros elementos e texto)
 - ... note-se que cada elemento pode sempre conter atributos
- As 4 formas de Tipo Complexo,
 - são definições que geralmente se utilizam
 - convém conhecê-las pois representam construções estereotipadas

Tipo Complexo: elemento-vazio

Um elemento-vazio apenas pode conter atributos.

```
<produto codigo="1234"/>
```

recordar: indica que se pretende estender ou **restringir** um tipo base

... ou, de modo mais compacto

```
<xs:element name="produto">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:anyType">
        <xs:attribute name="codigo"
          type="xs:positiveInteger"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="produto"/>
<xs:complexType>
  <xs:attribute name="codigo"
    type="xs:positiveInteger"/>
</xs:complexType>
</xs:element>
```

Técnica:
define-se um tipo que pode conter elementos,
mas não se declaram quaisquer elementos!

Tipo Complexo: elemento-texto

Um elemento-texto apenas pode conter texto (e naturalmente atributos).

```
<codReceita origem="Japão">15</codReceita>
```

```
<xs:element name="codReceita">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute name="origem"
          type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

simpleContent permite:
estender ou **restringir** um elemento-texto
ou
estender ou restringir o conteúdo um tipo simples
(aquele que não pode conter atributos; aquele
que apenas pode conter texto)

Técnica:
define-se um tipo complexo e diz-se que o seu
conteúdo é o de um tipo simples com atributos!

Tipo Complexo: elemento-elementos

Um elemento-elementos apenas pode conter outros elementos (e atributos).

```
<receita>
  <nomeOriginal>Sumeshi</nomeOriginal>
  <nomeUsual>Arroz de Sushi</nomeUsual>
</receita>
```

recordar: exemplo anterior

```
<xs:element name="receita">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nomeOriginal"
        type="xs:string"/>
      <xs:element name="nomeUsual"
        type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

sequence permite:
indicar a ordem pela qual os elementos surgem
no contexto do complexType

no exemplo:
primeiro surge o elemento nomeOriginal
depois surge o elementos nomeUsual

Técnica:
define-se um tipo que pode conter outros
elementos, em determinada sequência!

Tipo Complexo: elemento-misto

Um elemento-misto pode conter atributos, elementos e texto.

```
<receita>
  O
  <nomeOriginal>Sumeshi</nomeOriginal> é usualmente designado por
  <nomeUsual>Arroz de Sushi</nomeUsual>. Esta receita tem código
  <codReceita>15</codReceita>.
</receita>
```

O conteúdo de `receita` contém texto analisado (“parsed character data”, #PCDATA) intercalado com outros elementos.

```
<xs:element name="receita">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="nomeOriginal" type="xs:string"/>
      <xs:element name="nomeUsual" type="xs:string"/>
      <xs:element name="codReceita" type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

`mixed="true"` permite: colocar texto (#PCDATA) entre os elementos de um `complexType`

no exemplo:

“O”, “é usualmente designado por”, “. Esta receita tem código”, e “.”

são textos colocados entre os elementos descendentes de `receita`.

Tipo Complexo: indicador

- Um indicador define-se para um Tipo Complexo e permite
 - controlar a utilização dos elementos em documentos instância
 - existem 7 indicadores (organizados em 3 grupos)
- Indicador de relação de ordem
 - `all`, `choice`, `sequence`
- Indicador de ocorrência
 - `maxOccurs`, `minOccurs`
- Indicador de agrupamento
 - ◊ `group name`, `attributeGroup name`

Indicador de relação de ordem (`all`) e ocorrência

`all`

elementos podem surgir em qualquer ordem;
cada elemento apenas pode surgir uma vez.

`minOccurs`

pode ter valor 0 ou 1

`maxOccurs`

apenas pode ter valor 1

omissão = 1 (para ambos)

```
<xs:element name="receita">
  <xs:complexType>
    <xs:all>
      <xs:element name="nomeOriginal" type="xs:string"/>
      <xs:element name="nomeUsual" type="xs:string" minOccurs="0"/>
      <xs:element name="codReceita" type="xs:integer"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Indicador de relação de ordem (`choice`) e ocorrência

`choice`

apenas pode surgir um dos elementos indicados;
esse elemento pode surgir diversas vezes.

$0 \leq \text{minOccurs}$

$0 \leq \text{maxOccurs} \leq$
`unbounded` \equiv **sem limite**

omissão = 1 (para ambos)

```
<xs:element name="receita">
  <xs:complexType>
    <xs:choice>
      <xs:element name="numeroBilheteIdentidade" type="xs:string"/>
      <xs:element name="numeroInformacaoFiscal" type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Indicador de relação de ordem (sequence) e ocorrência

sequence

elementos surgem na ordem definida;
cada elemento pode surgir diversas vezes.

$0 \leq \text{minOccurs}$

$0 \leq \text{maxOccurs} \leq$
unbounded \equiv sem limite

omissão = 1 (para ambos)

```
<xs:element name="receita">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nomeOriginal" type="xs:string"/>
      <xs:element name="nomeUsual" type="xs:string" minOccurs="0"/>
      <xs:element name="codReceita" type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

... indicadores XSD e construções DTD

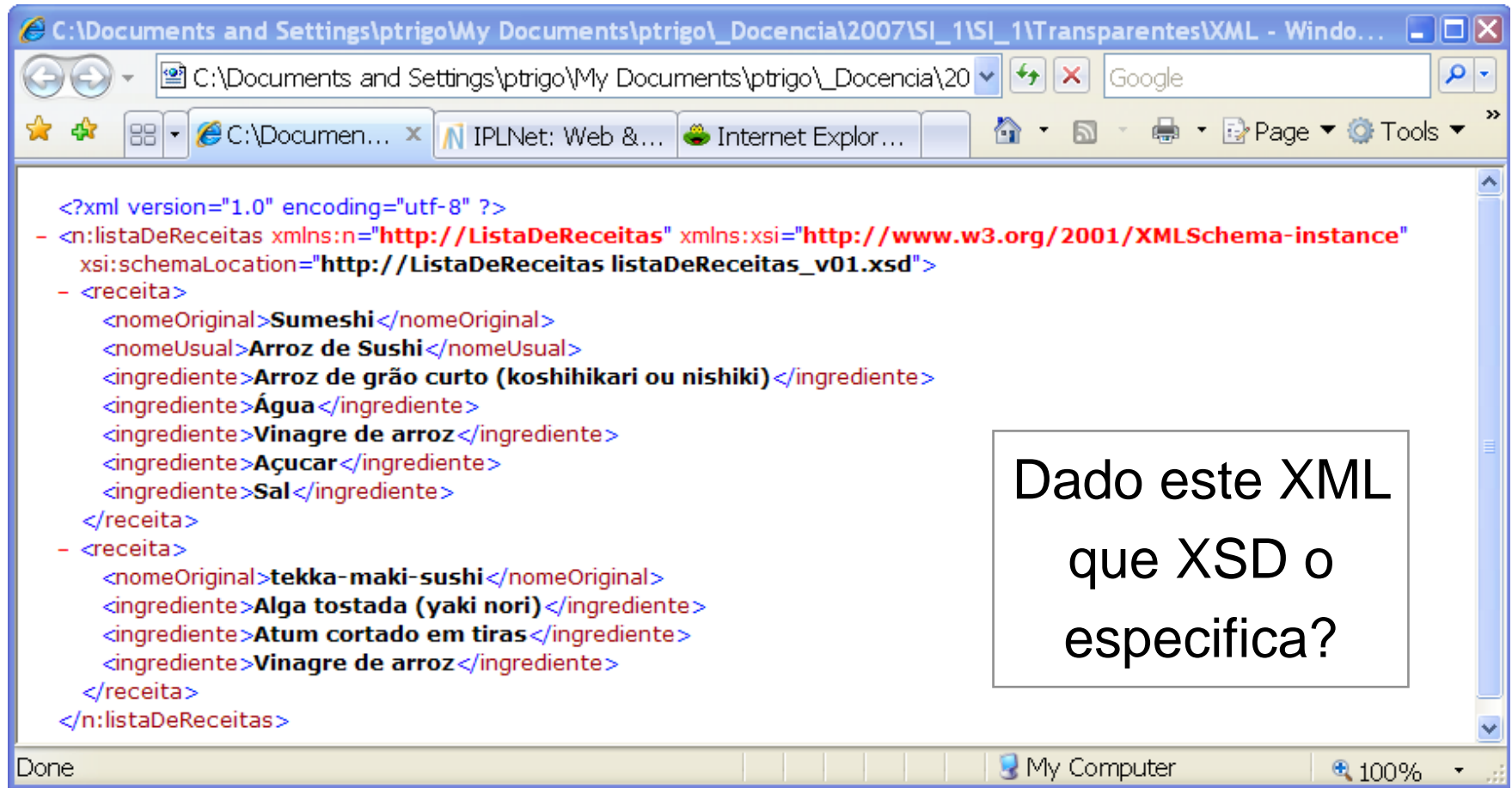
Correspondência entre indicadores XSD
e construções DTD (“Document Type Definition”)

XSD	DTD	Descrição
<code>xs:sequence</code>	elementos separados por vírgula, e.g. (e1, e2, e3)	Sequência ordenada de elementos.
<code>xs:choice</code>	elementos separados por barra, e.g. (e1 e2 e3)	Escolha de um dos elementos.
<code>xs:all</code>	não há equivalente	Qualquer dos elementos numa qualquer ordem.

Exemplo: lista de receitas

Uma lista de receitas.

Cada receita com o seu nome original e usual e lista de ingredientes.



```
<?xml version="1.0" encoding="utf-8" ?>
- <n:listaDeReceitas xmlns:n="http://ListaDeReceitas" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ListaDeReceitas listaDeReceitas_v01.xsd">
- <receita>
  <nomeOriginal>Sumeshi</nomeOriginal>
  <nomeUsual>Arroz de Sushi</nomeUsual>
  <ingrediente>Arroz de grão curto (koshihikari ou nishiki)</ingrediente>
  <ingrediente>Água</ingrediente>
  <ingrediente>Vinagre de arroz</ingrediente>
  <ingrediente>Açúcar</ingrediente>
  <ingrediente>Sal</ingrediente>
</receita>
- <receita>
  <nomeOriginal>tekka-maki-sushi</nomeOriginal>
  <ingrediente>Alga tostada (yaki nori)</ingrediente>
  <ingrediente>Atum cortado em tiras</ingrediente>
  <ingrediente>Vinagre de arroz</ingrediente>
</receita>
</n:listaDeReceitas>
```

Dado este XML
que XSD o
especifica?

Exemplo: lista de receitas (de novo o XML)

```
<listaDeReceitas>
```

```
  <receita>
```

```
    <nomeOriginal>Sumeshi</nomeOriginal>
```

```
    <nomeUsual>Arroz de Sushi</nomeUsual>
```

```
    <ingrediente>Arroz de grão curto (koshihikari ou nishiki)</ingrediente>
```

```
    <ingrediente>Água</ingrediente>
```

```
    <ingrediente>Vinagre de arroz</ingrediente>
```

```
    <ingrediente>Açúcar</ingrediente>
```

```
    <ingrediente>Sal</ingrediente>
```

```
  </receita>
```

```
  <receita>
```

```
    <nomeOriginal>tekka-maki-sushi</nomeOriginal>
```

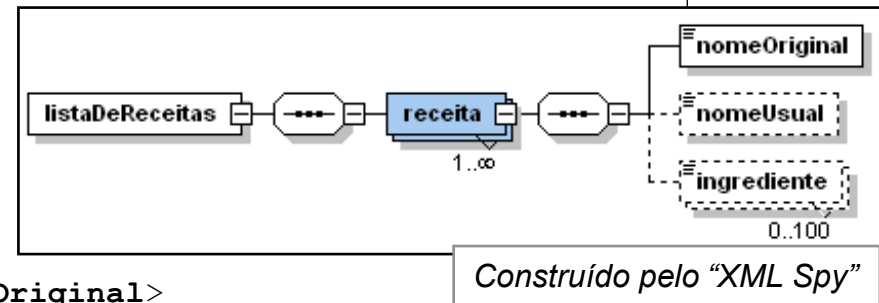
```
    <ingrediente>Alga tostada (yaki nori)</ingrediente>
```

```
    <ingrediente>Atum cortado em tiras</ingrediente>
```

```
    <ingrediente>Vinagre de arroz</ingrediente>
```

```
  </receita>
```

```
</listaDeReceitas>
```



listaDeReceitas.xml

Exemplo: lista de receitas (agora o XSD)

```
<xs:element name="listaDeReceitas">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="receita" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="nomeOriginal" type="xs:string"/>
            <xs:element name="nomeUsual" type="xs:string"
              minOccurs="0"/>
            <xs:element name="ingrediente" type="xs:string"
              minOccurs="0" maxOccurs="100"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

e.g. dois grupos:

1. nome original e usual, e
2. ingredientes.

... propor outra solução,
usando group name ...

listaDeReceitas_v01.xsd

Indicador de agrupamento (sobre elementos)

```
<xs:group name="grNomeDaReceita">
  <xs:sequence>
    <xs:element name="nomeOriginal" type="xs:string"/>
    <xs:element name="nomeUsual" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:group>
```

Declaração do
grupo de
elementos.

```
<xs:element name="listaDeReceitas">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="receita" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:group ref="grNomeDaReceita"/>
            <xs:element name="ingrediente" type="xs:string"
              minOccurs="0" maxOccurs="100"/>
          </xs:sequence>
        </... fechar restantes marcas
```

Utilização
do grupo.

listaDeReceitas_v02.xsd

Indicador de agrupamento (sobre atributos)

Uma pessoa, e.g. aluno ou docente, tem BI, nome e data de nascimento.

```
<xs:attributeGroup name="attrGrPESSOA">
  <xs:sequence>
    <xs:attribute name="nBI" type="xs:string"/>
    <xs:attribute name="nome" type="xs:string"/>
    <xs:attribute name="dataNasc" type="xs:date"/>
  </xs:sequence>
</xs:attributeGroup>
```

Declaração do
grupo de
atributos.

```
<xs:complexType name="tPESSOA">
  <xs:sequence>
    <xs:group ref="attrGrPESSOA"/>
  </xs:sequence>
</xs:complexType>
```

Utilização
do grupo.

```
<xs:element name="aluno" type="tPESSOA"/>
<xs:element name="docente" type="tPESSOA"/>
```

... o aluno e o docente.

Elementos não especificados (any)

O elemento `<any>` permite estender um documento XML com elementos não especificados no esquema (XSD).

```
<xs:element name="pessoa">
```

```
  <xs:complexType>
```

```
    <xs:sequence>
```

```
      <xs:element name="nBI" type="xs:string"/>
```

```
      <xs:element name="nome" type="xs:string"/>
```

```
      <xs:element name="dataNasc" type="xs:date"/>
```

```
      <xs:any minOccurs="0"/>
```

```
    </xs:sequence>
```

```
  </xs:complexType>
```

```
</xs:element>
```

a.xsd

```
<xs:element name="numTelef"/>
```

```
<xs:element name="morada"/>
```

b.xsd

```
<pessoa>
```

```
  <nBI>1234567</nBI>
```

```
  <nome>Yamoto</nome>
```

```
  <dataNasc>2000-01-31</dataNasc>
```

```
  <numTelef>910000019</numTelef>
```

```
  <morada>Rua X</morada>
```

```
</pessoa>
```

Atributos não especificados (anyAttribute)

O elemento `<anyAttribute>` permite estender um documento XML com atributos não especificados no esquema (XSD).

```
<xs:element name="pessoa">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nBI" type="xs:string"/>
      <xs:element name="nome" type="xs:string"/>
    </xs:sequence>
    <xs:anyAttribute/>
  </xs:complexType>
</xs:element>
```

a.xsd

```
<xs:attribute name="genero">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="masc|femin"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
```

b.xsd

```
<pessoa genero="femin">
  <nBI>1234567</nBI>
  <nome>Yamoto</nome>
</pessoa>
```

Substituição de elementos

A substituição de elementos permite, por exemplo, definir elementos com nomes em várias línguas.

Define-se um elemento `x` e depois definem-se os outros elementos que são substituíveis por esse elemento `x`.

```
<xs:element name="morada" type="xs:string"/>
<xs:element name="address" substitutionGroup="morada"/>

<xs:complexType name="tPessoa">
  <xs:sequence>
    <xs:element ref="morada"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="aluno" type="tPessoa"/>
<xs:element name="student" substitutionGroup="aluno"/>
```

```
<aluno>
  <morada>Rua X</morada>
</aluno>
```

ambos XML são válidos

```
<student>
  <address>Rua X</address>
</student>
```


Substituição de elementos (tipo e definição global)

- Considera-se que o `substitutionGroup` é constituído por
 - o elemento a substituir (o primeiro definido) e os que o substituem
- Todos os elementos do `substitutionGroup`
 - têm que ter o mesmo tipo, ou
 - um tipo derivado do elemento a substituir (o primeiro definido)
- Os elementos do `substitutionGroup`
 - têm que ser todos declarados como elementos globais
- ... recordar: um elemento é global quando
 - é descendente imediato do elemento `schema`
- ... recordar: um elemento é local quando
 - é descendente imediato de um elemento diferente de `schema`

Derivação (aspectos avançados): `abstract`

- É possível derivar tipos, surgindo assim as propriedades clássicas
 - de tipo abstracto (não tem instâncias) e tipo final (não se deriva)
- `abstract` é um atributo de `<complexType>` e de `<element>`
 - quando tem valor `"true"`,
 - indica que o tipo, ou elemento, não podem ser usados directamente
 - ... só podem ser usados os seus derivados
- ... se `<complexType abstract="true" ...>`, então no XSD
 - os elementos e atributos apenas podem ser de tipos derivados daquele
- ... se `<element abstract="true" ...>`, então no XML
 - apenas os elementos do seu `substitutionGroup` podem ser usados
 - ... mas não aquele elemento `abstract`

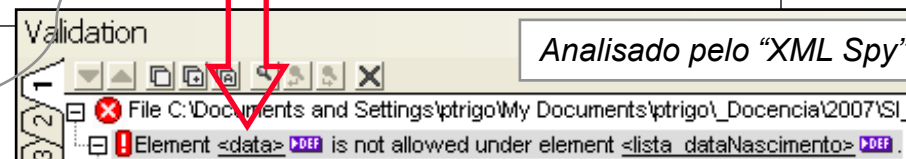
Exemplo: elemento abstract

```
<?xml version=""1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="data" type="xs:date" abstract="true"/>
  <xs:element name="dataNascimento" substitutionGroup="data"/>
  <xs:element name="lista_dataNascimento">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="data" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Exemplo de
derivação de
um elemento
abstracto.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<lista_dataNascimento xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="exemplo_abstract_v01.xsd">
  <dataNascimento>2007-01-31</dataNascimento>
  <data>2007-01-31</data>
</lista_dataNascimento>
```

Para estar correcto só pode ter
<dataNascimento> e nunca **<data>**



Derivação (aspectos avançados): `final`

- `final` é um atributo de `<complexType>` e de `<element>`
 - e pode ter valor: `extension`, `restriction` ou `#all`

<code>final</code>	<code><element></code>	<code><complexType></code>
<code>extension</code>	não permite estender	
<code>restriction</code>	não permite impor restrições	
<code>#all</code>	não permite qualquer derivação	

Exemplo: elemento com atributo final

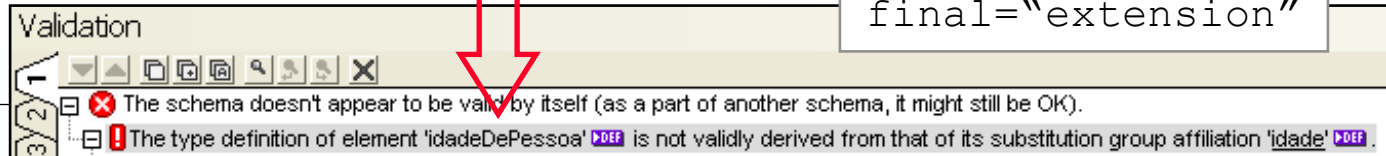
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="idade" type="xs:integer" final="restriction"/>
  <xs:element name="idadeDePessoa" substitutionGroup="idade">
    <xs:simpleType>
      <xs:restriction base="xs:integer">
        <xs:minInclusive value="0"/>
        <xs:maxInclusive value="120"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>

  <xs:element name="lista_idade">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="idade" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Exemplo de restrição de um elemento final.

Estaria correcto: sem atributo final, ou final="extension"



Restrições de integridade: é possível defini-las no XSD?

- Sim. O XSD permite impor, para o contexto de um documento XML,
 - unicidade dos valores de elementos e/ou de atributos
 - ◇ ... designado, no modelo relacional, por “integridade de entidade”
 - garantia dos valores de elementos e/ou atributos referirem outros
 - ◇ ... designado, no modelo relacional, por “integridade referencial”
- Unicidade dos valores de elementos e/ou atributos
 - o elemento `<unique>` impõe unicidade dos valores
 - ◇ ... corresponde, no modelo relacional, à “chave candidata”
 - o elemento `<key>` impõe unicidade e permite ser referenciado
 - ◇ ... corresponde, no modelo relacional, à “chave primária”
- Garantia dos valores de elementos e/ou atributos referirem outros
 - o elemento `<keyref>` impõe garantia de que o valor já exista
 - ◇ ... corresponde, no modelo relacional, à “chave estrangeira”

Unicidade dos valores de elementos e/ou atributos

- A construção da restrição de unicidade tem duas fases
 - 1º: definir a lista de elementos que devem ser avaliados
 - 2º: definir concretamente os elementos e/ou atributos de valor único
 - ... ambas as definições recorrem ao XPath
- Exemplo: as duas fases da construção da restrição de unicidade

```
<xs:element name="lista_bar" type="CT_lista_bar">  
  
  <xs:unique name="cc_codigo">  
    <xs:selector xpath="bar"/>  
    <xs:field xpath="@codigo"/>  
  </xs:unique>  
  
</xs:element>
```

O contexto que tem elementos e/ou atributos de valor único

um tipo `lista_bar` (a ver em seguida)

1º: a lista de elementos a analisar

2º: o atributo de valor único

Exemplo: uma lista de bares e o seu documento XML

Na lista de bares existem vários bares.

Cada bar tem um código único e que para simplificar admitimos ser o nome do bar.

Cada bar tem também uma morada que é única.

... um documento XML com a lista de bares

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>

<lista_bar xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="listaDeBares_v01.xsd">

  <bar codigo="O Bar Inglês">
    <morada>Rua Aru</morada>
  </bar>

  <bar codigo="O Pinguim">
    <morada>Rua Ura</morada>
  </bar>

</lista_bar>
```



listaDeBares_v01.xml

Exemplo: a estrutura XSD (... mas falta a unicidade!)

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:complexType name="CT_lista_bar">
    <xs:sequence>
      <xs:element ref="bar" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="CT_bar">
    <xs:sequence>
      <xs:element ref="morada"/>
    </xs:sequence>
    <xs:attribute name="codigo" type="xs:string"/>
  </xs:complexType>

  <xs:element name="bar" type="CT_bar"/>
  <xs:element name="morada" type="xs:string"/>

  ... (continua na próxima folha)
```

listaDeBares_v01.xsd

... e agora como
definir o elemento
lista_bar?

É preciso garantir
a unicidade do
atributo `codigo` e
do elemento
`morada`.

Exemplo: a especificação da unicidade

... (continuação da folha anterior)

```
<xs:element name="lista_bar" type="CT_lista_bar">
```

```
  <xs:unique name="cc_codigo">
```

```
    <xs:selector xpath="bar"/>
```

```
    <xs:field xpath="@codigo"/>
```

```
  </xs:unique>
```

```
  <xs:unique name="cc_morada">
```

```
    <xs:selector xpath="bar"/>
```

```
    <xs:field xpath="morada"/>
```

```
  </xs:unique>
```

```
</xs:element>
```

```
</xs:schema>
```

listaDeBares_v01.xsd

valor único

Exemplo: ... a validação da unicidade

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>

<lista_bar xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="listaDeBares_v01.xsd">

  <bar codigo="O Bar Inglês">
    <morada>Rua Aru</morada>
  </bar>

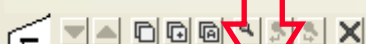
  <bar codigo="O Pinguim">
    <morada>Rua Aru</morada>
  </bar>

</lista_bar>
```

valor tem que ser único!

Analisado pelo "XML Spy"

Validation



- ❌ File C:\Documents and Settings\ptrigo\My Documents\ptrigo\Docencia\2007\SI_1\SI_1\Transparentes\XML_testes\05_XSD\listaDeBares_v02.xml is not valid.
- ❗ The value 'Rua Aru' was already matched by the <unique> identity constraint 'cc_morada' within the scope of element <lista_bar>.
- Details
 - cvc-identity-constraint.4.1: The value 'Rua Aru' was already matched by the <unique> identity constraint 'cc_morada' within the scope of element <lista_bar>.

Mas como garantir a integridade referencial?

- Definindo dois conceitos (*key*, ou *unique*, e *keyref*)
 - 1º: elemento e/ou atributo que se pode referenciar (*key* ou *unique*)
 - ◇ ... e.g. a “chave primária” (aquela eleita entre as candidatas!)
 - 2º: elemento e/ou atributo que refere uma chave (*keyref*)
 - ◇ ... a “chave estrangeira” (aquela que refere uma “chave primária”!)
- Exemplo: o atributo `codigo` foi eleito **chave** do bar!

```
<xs:element name="lista_bar" type="CT_lista_bar">  
  <xs:key name="cp_codigo">  
    <xs:selector xpath="bar"/>  
    <xs:field xpath="@codigo"/>  
  </xs:key>  
</xs:element>
```

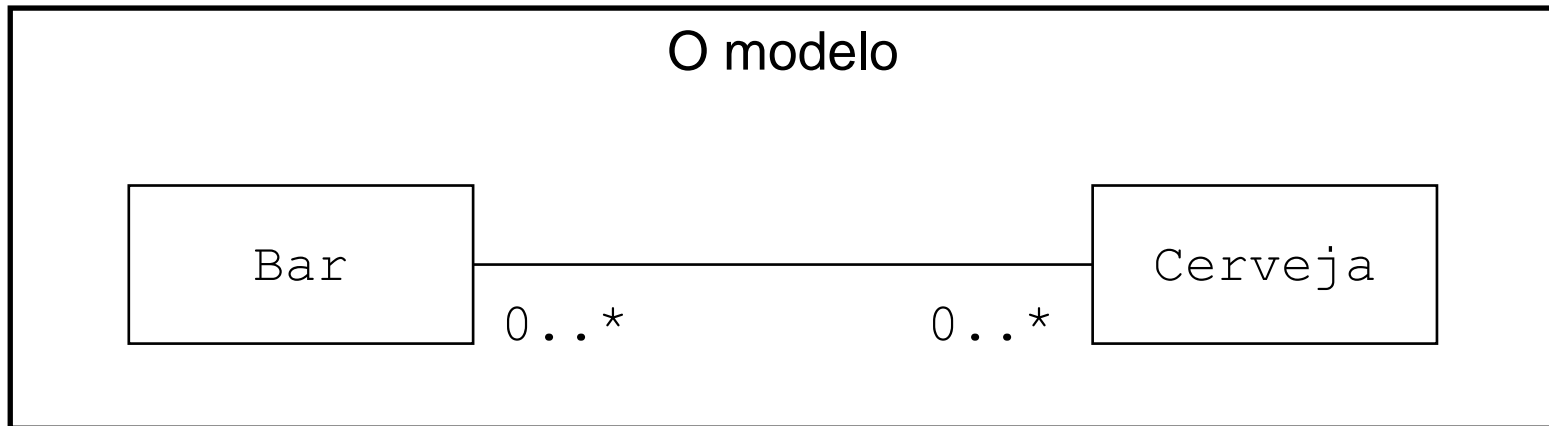
... em vez de
unique

Exemplo: outra lista de bares, mas com cervejas!

Agora pretendem-se três listas:

- a. uma lista com o nome dos bares,
- b. outra lista com o nome das cervejas, e
- c. outra lista que indique as cervejas de cada bar.

O modelo



Exemplo: a estrutura do bar e da cerveja

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:complexType name="CT_lista_bar_cerveja">
    <xs:sequence>
      <xs:element name="bar" type="xs:string" maxOccurs="unbounded"/>
      <xs:element name="cerveja" type="xs:string" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="lista_bar_cerveja" type="CT_lista_bar_cerveja">
    <xs:key name="cp_bar">
      <xs:selector xpath="bar"/>
      <xs:field xpath="."/>
    </xs:key>
    <xs:key name="cp_cerveja">
      <xs:selector xpath="cerveja"/>
      <xs:field xpath="."/>
    </xs:key>
  </xs:element>
</xs:schema>
```

→ chave de bar

→ chave de cerveja

... mas que bares têm que cervejas?

Para responder, definir elemento `bar_cerveja` que representa o produto cartesiano dos bares pelas cervejas (i.e. uma associação M:N)

... igual à folha anterior

```
<xs:complexType name="CT_lista_bar_cerveja">
```

```
<xs:sequence>
```

... igual à folha anterior

```
<xs:element name="bar_cerveja" maxOccurs="unbounded">
```

```
<xs:complexType>
```

```
<xs:attribute name="oBar" type="xs:string"/>
```

```
<xs:attribute name="aCerveja" type="xs:string"/>
```

```
</xs:complexType>
```

```
</xs:element>
```

```
</xs:sequence>
```

```
</xs:complexType>
```

... igual à folha anterior

O elemento
`bar_cerveja` tem
dois atributos:
`oBar` e `aCerveja`.

Aqueles atributos
terão o conteúdo,
respectivamente dos
elementos:
`bar` e `cerveja`.

... no entanto, estes bares e estas cervejas não existem!

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>

<lista_bar_cerveja xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="listaDeBares_v02.xsd">

    <bar>O Bar Inglês</bar>
    <bar>O Pinguim</bar>

    <cerveja>Antarctica</cerveja>
    <cerveja>Bud</cerveja>

    <bar_cerveja oBar="O Sushi Bar" aCerveja="Sapporo"/>
    <bar_cerveja oBar="O Belga" aCerveja="Hoegaarden"/>

</lista_bar_cerveja>
```

Não
constam
no
conteúdo
de bar
nem de
cerveja!

... garantir que os bares e as cervejas existem!

... igual à folha anterior

```
<xs:element name="lista_bar_cerveja" type="CT_lista_bar_cerveja">
  <xs:key name="cp_bar">
    <xs:selector xpath="bar"/> <xs:field xpath="."/>
  </xs:key>
  <xs:key name="cp_cerveja">
    <xs:selector xpath="cerveja"/> <xs:field xpath="."/>
  </xs:key>
  <xs:keyref name="ce_01_bar_cerveja" refer="cp_bar">
    <xs:selector xpath="bar_cerveja"/>
    <xs:field xpath="@oBar"/>
  </xs:keyref>
  <xs:keyref name="ce_02_bar_cerveja" refer="cp_cerveja">
    <xs:selector xpath="bar_cerveja"/>
    <xs:field xpath="@aCerveja"/>
  </xs:keyref>
</xs:element>
```

... igual à folha anterior

Também estaria
correcto `unique`
em vez de `key`

referência
para um valor
de `cp_bar`

referência para
um valor de
`cp_cerveja`

Exemplo: ... a validação da integridade referencial

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>

<lista_bar_cerveja xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="listaDeBares_v04.xsd">

  <bar>O Bar Inglês</bar>
  <bar>O Pinguim</bar>

  <cerveja>Antarctica</cerveja>
  <cerveja>Bud</cerveja>

  <bar_cerveja oBar="O Bar Inglês" aCerveja="Sagres"/>
  <bar_cerveja oBar="O Pinguim" aCerveja="Antarctica"/>

</lista_bar_cerveja>
```

valor tem que existir
no elemento bar!

Analisado pelo "XML Spy"

Validation

File C:\Documents and Settings\ptrigo\My Documents\ptrigo\Docencia\2007\SI_1\SI_1\Transparentes\XML_testes\05_XSD\listaDeBares_v04.xml is not valid.

The value 'Sagres' matched by the <keyref> identity constraint 'ce_02_bar_cerveja' was not matched by the referenced key 'cp_cerveja' within the scope of element <lista_bar_cerveja>.

Error location: lista_bar_cerveja / bar_cerveja / @aCerveja

Details

cvc-identity-constraint.4.3: The value 'Sagres' matched by the <keyref> identity constraint 'ce_02_bar_cerveja' was not matched by the referenced key 'cp_cerveja' within the scope of elei

Resumo: unique, key e keyref

- unique
 - elemento ou atributo cujo valor tem que ser único
- key
 - elemento ou atributo cujo valor tem que ser único (tal como unique), e
 - tem que estar sempre presente (no contexto em que está definido)
- keyref
 - elemento ou atributo cujo valor existe no key ou unique referenciado
- Integridade referencial XML é mais geral que no modelo relacional
 - keyref pode referir elementos/atributos key ou unique
 - ◇ ... no modelo relacional, a chave estrangeira
 - ◇ só pode referenciar uma chave primária (não uma candidata)
- No entanto, é boa prática keyref apenas referenciar key!

Um único esquema separado em diversos documentos

- A boa prática da reutilização sugere construir documentos XSD
 - que possam ser os “módulos” da construção de outros esquemas
- Compor um esquema a partir de outros recorre aos elementos
 - `xs:include`, `xs:redefine` ou `xs:import`
- **<xs:include>** inclui conteúdo com igual “target namespace” (*TN*)

```
<xs:include schemaLocation="URI"/>
```

- **<xs:redefine>** é como `<xs:include>` mas permite redefinição

```
<xs:redefine schemaLocation="URI"/>
```

- **<xs:import>** é como `<xs:include>` mas permite diferentes *TN*

```
<xs:import namespace="URI" schemaLocation="URI"/>
```

Uma estrutura a reutilizar (a incluir noutros contextos)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://ListaDeBarCerveja"
    attributeFormDefault="unqualified"
    elementFormDefault="unqualified">
  <xs:complexType name="CT_lista_bar_cerveja">
    <xs:sequence>
      <xs:element name="bar" type="xs:string" maxOccurs="unbounded"/>
      <xs:element name="cerveja" type="xs:string" maxOccurs="unbounded"/>
      <xs:element name="bar_cerveja" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="oBar" type="xs:string"/>
          <xs:attribute name="aCerveja" type="xs:string"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Documento XML pode usar estes atributos e elementos sem os qualificar.

O tipo CT_lista_bar_cerveja será reutilizado ...

listaDeBares_include_v01_CT.xsd

... incluir aquela estrutura: <xs:include>

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://ListaDeBarCerveja"
  xmlns:iLBC="http://ListaDeBarCerveja"
  attributeFormDefault="unqualified" elementFormDefault="unqualified">
```

... mesmo "targetNamespace" que o do XSD incluído, e ... é preciso qualificar os nomes desse "namespace"

```
<xs:include schemaLocation="listaDeBares_include_v01_CT.xsd"/>
```

... incluir o XSD da folha anterior!

```
<xs:element name="lista_bar_cerveja" type="iLBC:CT_lista_bar_cerveja">
  <xs:key name="cp_bar"> <xs:selector xpath="bar"/> <xs:field xpath="."/> </xs:key>
  <xs:key name="cp_cerveja"> <xs:selector xpath="cerveja"/> <xs:field xpath="."/> </xs:key>
  <xs:key name="cp_bar_cerveja"> <xs:selector xpath="bar_cerveja"/>
    <xs:field xpath="@oBar"/> <xs:field xpath="@aCerveja"/> </xs:key>
  <xs:keyref name="ce_01_bar_cerveja" refer="iLBC:cp_bar">
    <xs:selector xpath="bar_cerveja"/> <xs:field xpath="@oBar"/> </xs:keyref>
  <xs:keyref name="ce_02_bar_cerveja" refer="iLBC:cp_cerveja">
    <xs:selector xpath="bar_cerveja"/> <xs:field xpath="@aCerveja"/> </xs:keyref>
</xs:element>
</xs:schema>
```

Não é nome local; é nome do "namespace" http://ListaDeBarCerveja

listaDeBares_include_v01.xsd

... o documento XML

Nota: apenas o elemento raiz está qualificado.

Não é preciso qualificar restantes elementos e atributos porque o XSD indica:

`attributeFormDefault="unqualified"` e `elementFormDefault="unqualified"`

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xLBC:lista_bar_cerveja
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ListaDeBarCerveja listaDeBares_include_v01.xsd"
  xmlns:xLBC="http://ListaDeBarCerveja">

  <bar>O Bar Inglês</bar>
  <bar>O Pinguim</bar>

  <cerveja>Antarctica</cerveja>
  <cerveja>Bud</cerveja>

  <bar_cerveja oBar="O Bar Inglês" aCerveja="Bud"/>
  <bar_cerveja oBar="O Pinguim" aCerveja="Antarctica"/>

</xLBC:lista_bar_cerveja>
```

O XSD pode obrigar, o XML, a qualificar os nomes?

Sim. Basta o XSD indicar:

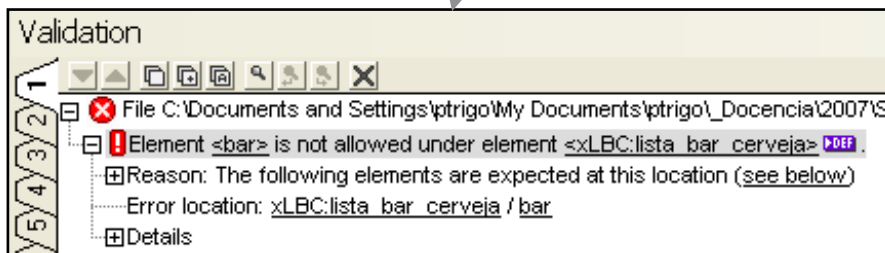
`attributeFormDefault="qualified" e/ou elementFormDefault="qualified"`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xLBC:lista_bar_cerveja
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ListaDeBarCerveja listaDeBares_include_v01.xsd"
  xmlns:xLBC="http://ListaDeBarCerveja">
```

```
<bar>O Bar Inglês</bar>
<bar>O Pinguim</bar>
<cerveja>Antarctica</cerveja>
<cerveja>Bud</cerveja>
<bar_cerveja oBar="O Bar Inglês" aCerveja="Bud"/>
<bar_cerveja oBar="O Pinguim" aCerveja="Antarctica"/>
```

```
</xLBC:lista_bar_cerveja>
```

Em `listaDeBares_include_v01_CT.xsd`,
(onde se define o tipo `CT_lista_bar_cerveja`)
obrigar a qualificar tem como efeito ...



Exemplo: obrigar a qualificar os elementos

Se em `listaDeBares_include_v01_CT.xsd`,
(*cf. folha anterior* onde se define o tipo `CT_lista_bar_cerveja`)
tiver `attributeFormDefault="unqualified"` e `elementFormDefault="qualified"`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xLBC:lista_bar_cerveja
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ListaDeBarCerveja listaDeBares_include_v01.xsd"
  xmlns:xLBC="http://ListaDeBarCerveja">

  <xLBC:bar>O Bar Inglês</xLBC:bar>
  <xLBC:bar>O Pinguim</xLBC:bar>
  <xLBC:cerveja>Antarctica</xLBC:cerveja>
  <xLBC:cerveja>Bud</xLBC:cerveja>
  <xLBC:bar_cerveja oBar="O Bar Inglês" aCerveja="Bud"/>
  <xLBC:bar_cerveja oBar="O Pinguim" aCerveja="Antarctica"/>

</xLBC:lista_bar_cerveja>
```

Elementos foram obrigados a estar qualificados (qualified).

Atributos não foram obrigados (unqualified).

Incluir e redefinir outra estrutura: `<xs:redefine>`

- O `<xs:redefine>` permite redefinir tipos simples e complexos
 - ... e também grupos de elementos e de atributos
- As componentes lidas têm que estar no mesmo “target namespace”
 - ou não estarem associadas a nenhum “target namespace”
- ... se as componentes não estão associadas a “target namespace”
 - passam a estar no “target namespace” do esquema que os redefine
- No contexto do `<xs:redefine>`, as componentes não redefinidas
 - são incluídas tal como em `<xs:include>`

Exemplo: redefinir tipo simples adicionando uma faceta

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://Exemplo_redefine"
  attributeFormDefault="qualified"
  elementFormDefault="qualified">

  <xs:simpleType name="ST_umTipoSimples">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
</xs:schema>
```

exemplo_redefine_v01_ST.xsd

O tipo simples

O tipo simples
redefinido

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://Exemplo_redefine"
  xmlns:rER="http://Exemplo_redefine"
  attributeFormDefault="qualified"
  elementFormDefault="qualified">

  <xs:redefine schemaLocation="exemplo_redefine_v01_ST.xsd">
    <xs:simpleType name="ST_umTipoSimples">
      <xs:restriction base="rER:ST_umTipoSimples">
        <xs:minLength value="5"/> <xs:maxLength value="8"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:redefine>
... continua na próxima folha
```

... poderia ser outro "namespace"!

... uma lista de “senhas”, cada uma com 5 a 8 caracteres

... continuação da folha anterior

```
<xs:element name="lista_senha">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="senha" type="rER:ST_umTipoSimples" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

exemplo_redefine_v01.xsd

Um documento XML que segue a especificação da “lista de senhas”

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xER:lista_senha xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://Exemplo_redefine exemplo_redefine_v01.xsd"
  xmlns:xER="http://Exemplo_redefine">
  <xER:senha>abcdef</xER:senha>
  <xER:senha>12345678</xER:senha>
</xER:lista_senha>
```

Estão qualificados porque o XSD assim obriga
(elementFormDefault="qualified")

Importar outra estrutura: `<xs:import>`

- O `<xs:import>` permite incorporar tipos e elementos globais
 - mesmo que de diferentes “target namespace”
- Recordar que `<xs:include>` e `<xs:redefine>` exigem
 - que os nomes pertençam ao mesmo “target namespace”

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://xxxxxxxxxx"
  attributeFormDefault="qualified"
  elementFormDefault="qualified">
  <xs:simpleType name="ST_umTipoSimples">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
</xs:schema>
```

exemplo_import_v01_ST.xsd

O tipo simples mas
agora para outro
espaço de nomes
(para outro “target
namespace”)

... importar esquema com outro espaço de nomes

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://EsteEspacoDeNomes"
  xmlns:Nseste="http://EsteEspacoDeNomes"
  xmlns:NSimportado="http://XXXXXXXXXX"
  attributeFormDefault="qualified" elementFormDefault="qualified">

  <xs:import namespace="http://XXXXXXXXXX" schemaLocation="exemplo_import_v01_ST.xsd"/>

  <xs:simpleType name="tSenha">
    <xs:restriction base="NSimportado:ST_umTipoSimples">
      <xs:minLength value="5"/> <xs:maxLength value="8"/>
    </xs:restriction> </xs:simpleType>

    <xs:element name="lista_senha">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="senha" type="NSeste:tSenha" maxOccurs="unbounded"/>
        </xs:sequence> </xs:complexType> </xs:element>
      </xs:schema>
```

O "target namespace" do esquema importado

... a mesma lista de senhas

... mas agora o esquema que o XML respeita importou outro esquema que por sua vez tinha um “target namespace” diferente!

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xEEN:lista_senha xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://EsteEspacoDeNomes exemplo_import_v01.xsd"
    xmlns:xEEN="http://EsteEspacoDeNomes">

    <xEEN:senha>abcdef</xEEN:senha>
    <xEEN:senha>12345678</xEEN:senha>

</xEEN:lista_senha>
```

Estão qualificados porque o XSD assim obriga
(elementFormDefault="qualified")

O que o XSD fornece e o que não fornece...

- O XSD permite especificar
 - tipos de dados simples e complexos
 - extensão de tipos
 - restrições sobre ocorrências de elementos
 - restrições sobre valores de elementos e de atributos
 - restrições sobre unicidade dos valores de elementos e atributos
 - restrições sobre referencias entre valores de elementos e atributos
 - especificação de estruturas compatíveis com espaços de nomes
- O XSD não permite especificar
 - entidades para utilizar em documentos XML
 - ... de modo tão simples como o DTD a estrutura do documento XML!
- ... por ser extremamente mais poderoso que DTD é mais complexo!

Qual o próximo passo?

Para especificar a estrutura de documentos XML
o XSD é actualmente o último passo!

O XSD é forte candidato a substituto do DTD
no suporte a aplicações Web.

Depois de especificar a estrutura é preciso navegar nos
dados e para isso é preciso avançar para o XQuery!

Tal como no SQL, depois de especificar o esquema relacional com a
LDD, é preciso avançar para manipular os dados com a LMD.